MAGATAR™

VOLUME 2 NUMBER 1

for the ATARI 400/800

1 DATA BOSS #3

2 6502 TUTORIAL #5

3 VARIABLE CHANGE UTILITY

4 PHAROAH'S CURSE ADVENTURE

5 DISASSEMBLER

$14.95 for disk, $12.95 for cassette

programmer's institute
a division of **FUTUREHOUSE™**
310½ West Franklin Street
P.O. Box 3470, Chapel Hill, NC 27514
1-800-334-SOFT
orderline

Editor's Notes

Keeping in line with the explosive growth in the microcomputer software industry today, we have been forced to move into larger quarters. We've moved twice just in the past year (some backs are permanently damaged) and now have about 3,000 square feet of working space. We're also pleased to announce some additions to our staff. John Watkin has joined us as our Director of Public Relations. John came to us as a freelance writer and is currently neck-deep in company promotion work. David Hock (yes, we are related) has come onboard as our Director of Research and Development. Dave has degrees in Electrical Engineering and Computer Science from SUNY at Stony Brook, NY. He has been helpful in the conversion of our Atari personal finance program to the Commodore computers.

Speaking of growing...Atari seems to be on the tail end of some growing pains. I say "tail end" because though the 1200XL was a dud (with its incompatible operating system and hardware problems and deficiencies) they are going through yet another management shuffle: they combined their video game and home computer divisions and tapped an Executive V. P. from Phillip Morris as their new President. Atari's new entries, the 600XL, 800XL, 1400XL, 1450XL, and 1450XLD, are absolutely fantastic. For instance, the 1450XLD will retail for under $1000, have a built in modem, speech synthesizer, double density disk drives, and best of all NO MORE NEED FOR AN INTERFACE MODULE! Of course, the software writers and buyers should be pleased also, because the operating systems on the new computers are completely compatible with the 400/800.

Speaking of Atari, it seems most of the industry opinions concerning the current hardware shakeout predict that the order will be IBM, Apple, Radio Shack! For machines priced under $1000, we think the list will look more like #1 IBM, #2 Commodore, with Atari running a close third. Everyone should know that APPLE machines are overpriced (they're selling on the black market in the Far East for under $300) and that RADIO SHACK doesn't support third party software developers.

Speaking of software, Digital Retailing will have a computer supplement in the October 31 issue of Newsweek. In the supplement, they will recommend that people planning on buying a microcomputer should decide what they want to use it for first. Then they should buy the computer that has the software available that is needed. In other words...buy the software first. With the wealth of software available for the Atari, it can't help but be a bestseller.

And speaking of bestsellers, we hope you saw our ad for the 1st Annual Hacker's Open in the August, 1983 issue of Compute! We've canvassed many user's groups, schools, etc, and have already received some submissions...but not many from our subscribers. Why? Unfortunately, we can't answer that question. Only you can. So please call us (1-800-334-SOFT) and let us know where all your programs are!

And speaking of questions, we have a survey this issue that we would like everyone to fill out and send back to us. What's so important about this survey? Well, we're expanding the format of MAGATAR. First, we'll be changing the name to LOADED! and increasing the number of programs to 12-15 per issue. We are also expanding the written section of the magazine to include articles on programming, educational computing, and program reviews. Please send us your input...meanwhile, enjoy!

Loading Instructions

Program Names	Cassette Counter #	Diskette Instructions
Magatar Menu		RUN"D:MAGATAR"
Databoss #3	17 10 - 159	RUN"D:DATAPRNT"
6502 Tutorial #5	163 159 - 174	RUN"D:MACHINE.5"
Variable Change Utility	177 174 - 187	RUN"D:VARCHANG"
Pharoah's Curse Adventure	190 187 - 241	RUN"D:PHAROAH"
Disassembler	244 241 - 266	RUN"D:DISASM"

Cassette: All programs are copied on both sides of the tapes. If you should have loading problems, please try the other side of the tape.

Turn your Atari computer on. Advance the tape recorder to the specified line counter and press the [play] button; type in CLOAD and press [return] twice. If you should receive an ERROR 138 or an ERROR 140, advance the tape a counter or two past the indicated number. If you get an ERROR 143, load the program a counter or two before the number suggested. It may require several attempts to find the exact spot on the tape before the program can actually be loaded. Please be patient. Since the Atari cassette recorder is so unpredictable, we suggest that you load the first program on the tape, jot down the line number, then load the second program, jot down the line number and so on. This way you will have your own accurate starting numbers.

Diskette: Turn your monitor on. Insert the Magatar diskette in your drive and turn it on. When the busy light goes out, turn your computer on, and the programs will be loaded through the auto-boot system. If you should get a BOOT ERROR, please turn your computer off and re-start the entire process. If an ERROR 144 or an ERROR 140 should appear on your screen more than once, have you computer or your disk drive checked. If any one of these errors should persist after several trials, please return the disk for a prompt replacement. To load the programs separately, enter the name listed under the instructions, and press [return].

The Databoss #3

Finally, we have added the third module to the DATABOSS series. In this issue, we have included DATAPRINT, a utility that allows you to select records in a file to print; design a report; add/subtract/multiply/divide fields, and create new fields to show these results. You can even print totals, or an average for any field.

The main menu of DATAPRINT will look like this:

```
                     THE DATABOSS   [VER. 1. 1]
                      DATA SELECT/PRINT MODULE
                  copyright (c) 1983 by FUTUREHOUSE

                  [1]  load file
                  [2]  disk directory
                  [3]  design report
                  [4]  select/print
                  [5]  return to DATABOSS

                     What is your choice?
```

To load a DATABOSS file select option 1. The following will appear:

```
                     THE DATABOSS -- FILE MODE
```

Press [M]enu or [RETURN] to continue.

To load a file, press [RETURN]. The next screen will appear:

```
                   THE DATABOSS -- FILE MODE

            enter filename for application
            DISK FORMAT WILL BE "D: filename.DAT"

                         [        ]
```

Enter the name of the DATABOSS file you wish to load. Then press [RETURN] if the filename is less than 8 characters long.

The following prompt will appear:

```
               INSERT DATAMASTER DISKETTE IN DRIVE 1

                     press any key to continue
```

After satisfying this prompt by inserting your DATAMASTER in the drive, you will be informed that the program is now loading the DATABOSS. When the master file is loaded, you will be asked to insert your first data diskette in the drive. This will always be the same diskette as the DATAMASTER diskette, unless the file is more than one diskette long.

After pressing a key, you will momentarily return to the main menu.

If you wish to see a directory of the files on a disk, insert the disk in the drive, and select option 2 for disk directory.

Before you can print a report, you must first design one. To do this, select option 3 from the main menu. The following menu will appear:

```
                 THE DATABOSS -- REPORT DESIGN AID
```

```
                .........1.........2.........3.........4
                .........5.........6.........7.........8
                .........1.........2.........3.........4
                .........5.........6.........7.........8
                .........1.........2.........3.........4
                .........5.........6.........7.........8
                .........1.........2.........3.........4
                .........5.........6.........7.........8
                .........1.........2.........3.........4
                .........5.........6.........7.........8

      fld#   name   length              fld#   name   length
      [ ] [        ] [ ]                [ ] [        ] [ ]

              [H]eadings   [L]oad   [S]ave   [M]enu   [U]p
               [D]own   [A]ssign fields   [F]ormat report
```

When you design a report, you must complete 3 steps. First, you must print headings for the fields you want to print. They do not have to be the same as the headings for the fields in the DATABOSS file.

Next, we must assign fields to each of the field headings you printed. Then we must format how long we want each field to be and if it is to be formatted in dollars, in numerals, in integers, or in alphanumeric characters.

To design headings select [H]. A blinking cursor will appear at the top left corner of the screen. Since each line of headings will be 80 columns wide on your printer, two lines on the screen correspond to each printer line. To print your headings, you type exactly how you want them to look on your report, except for the following:

Before typing in each field name hit [CONTROL] and [L].

After typing in each field name, type in [CONTROL] and [R].

If you want two field names to be only 1 space apart then hit [CONTROL] and [B] instead of [CONTROL] [L] and [CONTROL] [R]. When finished typing in your report field title names, hit [RETURN].

Let's use our home inventory file as an example. The file looks like this:

TITLE	LENGTH	TYPE
1 NAME OF ITEM	25 char	alphanumeric
2 CONDITION	2 char	alphanumeric
3 QUANTITY	3 char	numeric
4 PRICE PAID	10 char	dollars/cents
5 PRESENT VALUE	10 char	dollars/cents

If we wanted a report that printed the NAME OF ITEM, the QUANTITY, PRICE PAID, and PRICE PER ITEM, we would type in the following:

```
...FURNITURE.........QUANTITY3.......TOT
AL..PRICE5.PAID....6....PRICE7.PER.ITEM8
.........1.........2.........3.........4
.........5.........6.........7.........8
.........1.........2.........3.........4
.........5.........6.........7.........8
.........1.........2.........3.........4
.........5.........6.........7.........8
.........1.........2.........3.........4
.........5.........6.........7.........8

  fld#      name     length                fld#    name    length
  [ 1] [name of item] [25]                 [ 2] [condition] [ 2]

          [H]eadings   [L]oad   [S]ave   [M]enu   [U]p
            [D]own   [A]ssign fields   [F]ormat report
```

The above is how your report design aid screen will look after typing out the title fields for your home inventory report.

Our next step is to assign the fields from our home inventory file to the fields in the report. Press [A] to begin assigning fields. You will be informed to PLEASE WAIT. Momentarily, the following screen will appear:

```
              THE DATABOSS -- REPORT DESIGN AID

        ...FURNITURE.........QUANTITY3.......TOT
        AL..PRICE5.PAID....6....PRICE7.PER.ITEM8
        .........1.........2.........3.........4
        .........5.........6.........7.........8
        .........1.........2.........3.........4
        .........5.........6.........7.........8
        .........1.........2.........3.........4
        .........5.........6.........7.........8
        .........1.........2.........3.........4
        .........5.........6.........7.........8

  fld#      name     length                fld#    name    length
  [ 1] [NAME OF ITEM] [25]                 [ 2] [CONDITION] [ 2]

 RF#      FF NAME      FF#     [+-*/0]     FF#     S/A/N
[ 1]  [            ] [  ]     [   ]      [  ]      [ ]

        [E]dit   [N]ext   [L]ast   [M]enu   [U]p   [D]own
```

To assign fields, select [E]dit. A blinking cursor will appear under the FF# (file field #). Since the report field title FURNITURE is to contain the NAME OF ITEM data, we want to type field #1 in the FF#. Next, the blinking cursor moves to the [+-*/0] field. Here, the program is asking you if you wish to add, subtract, multiply, or divide this field by another field in the file. Since we don't want to do this, we select 0.

The next FF# is only used if we select +-*/ in the prior prompt. Simply type 00 to pass this prompt. The last prompt asks if we want totals printed (S), an average printed (A), or no totals printed (N). Since we don't want any totals printed, select (N).

To select the file field # to assosciate with the next report field #, select [N]ext to access the next report field. Then select [E]. Continue this process until you have associated all the report fields with a file field.

The information for the rest of the report fields should contain the following:

RF#	FF Name	FF#	[+-*/0]	FF#	S/A/N
2	QUANTITY	3	0	0	S
3	PRICE PAID	4	0	0	S
4	PRICE PAID	4	/	3	A

To return to the prior list of options select [M]enu. The following screen will appear:

```
              THE DATABOSS -- REPORT DESIGN AID

            ...FURNITURE.........QUANTITY3.......TOT
            AL..PRICE5.PAID....6... PRICE7.PER.ITEM8
            .........1.........2.........3.........4
            .........5.........6.........7.........8
            .........1.........2.........3.........4
            .........5.........6.........7.........8
            .........1.........2.........3.........4
            .........5.........6.........7.........8
            .........1.........2.........3.........4
            .........5.........6.........7.........8

   fld#     name      length                fld#    name     length
   [ 1] [name of item] [25]                 [ 2] [condition] [ 2]

          [H]eadings   [L]oad   [S]ave   [M]enu   [U]p
           [D]own   [A]ssign fields   [F]ormat report
```

Last, we must format our report. When formatting our report, we decide how many characters in each field we want to print, and if we want to print the fields alphanumerically, numerically, in integer format, or in dollars/cents format. To format the report, select option [F]ormat report. The blinking cursor will again appear at the top of the screen. Here's what the screen should look like when you are done formatting:

```
              THE DATABOSS -- REPORT DESIGN AID

            .. AAAAAAAAAAE....IIIE...3...........8
            DDDDDDDDE.........6...DDDDDDDDE......8
            .........1.........2.........3.........4
            .........5.........6.........7.........8
            .........1.........2.........3.........4
            .........5.........6.........7.........8
```

```
                .........1.........2.........3.........4
                .........5.........6.........7.........8
                .........1.........2.........3.........4
                .........5.........6.........7.........8

      fld#     name     length                  fld#    name      length
      [ 1] [name of item] [25]                  [ 2] [condition] [ 2]

               [H]eadings   [L]oad   [S]ave   [M]enu   [U]p
                 [D]own   [A]ssign fields   [F]ormat report
```

Even though the length of the NAME OF ITEM field is 25 characters long, we only want to print the first 12 in our report. Notice that we use A for alphanumeric, N for numeric, I for integer, and D for dollars. Also, we placed an "E" at the end of each field to let the program know where each field ends.

Now we're ready to print our report. Select [M]enu to return to the main menu.

Select option 4 to print the report. The following screen will appear:

```
                  THE DATABOSS - PRINT/SELECT MODE

REC#0
                              NAME OF ITEM
                     [                          ]

          CONDITION                                  QUANTITY
            [  ]                                       [   ]

          PRICE PAID                               PRESENT VALUE
         [         ]                                 [         ]

               [R]ecord#   [C]urrent   [F]rom/to   [A]ll
              [M]enu   [N]ext   [L]ast   [S]elect from/to
```

We can see individual records by selecting [N]ext record, [L]ast record, or [R]ecord number. To print the current record select [C]urrent. To print all the records in the file, select [A]ll records. To print all records within a certain range, select [S]elect from/to.

When done, select [M]enu to return to the main menu.

6502 Tutorial #5

In our last tutorial we discussed the features of CIO, and gave some examples on how to use CIO to perform graphics functions normally executed from BASIC such as a GRAPHICS 0 or GRAPHICS 2 + 16, DRAW, and PLOT. We also talked about the many other features of CIO...its ability to print to any device, whether it's the screen, the printer, the disk, etc. It appears that CIO is the best way to perform graphics, right?

Wrong! The problem with using CIO to execute graphics is that it is a generalized routine that must be able to do more than just screen graphics. It has to check to see if you're opening the screen or printer, if the device is open for input or output or both...so if we can set up our own graphics screen, and our own plot routine, we don't have to use CIO and we will be able to execute graphics at a much greater speed.

But what does CIO do when it executes, say, a GRAPHICS 0? In our last issue we explained its procedures from a BASIC point of view. Now, we'll explain it from a machine language programmer's perspective.

First, we know from our 6502 Tutorial #2 about display lists, display list interrupts, and the function of the ANTIC chip. If we want to set up our own screen, all we have to do is set up a display list, and tell ANTIC where our display list is located. This way we can have our own screens. Unfortunately, if we set up our own display list, we won't be able to DRAW and PLOT to this screen because we bypassed CIO (we never "OPEN"ed an ICOB to print to the screen). So next we have to come up with our own plotting routine.

The routine we'll use is called bit mapping. Bit mapping is where we take a shape that we have plotted onto the screen and move the shape to a new location. For instance, a GRAPHICS 8 screen is 40 bytes wide and 192 bytes long (40 X 192 = 7680 bytes of RAM). So if we want to move a shape down one line, we must add 40 bytes to the current location of each vertical line of the shape. To move the shape horizontally we have to add or subtract one byte (a more complicated way of moving the shape is to move it left or right a BIT at a time, but more on this later). We'll use a shape table to define the shape(s) we want to plot and move on the screen.

First let's show you how to set up the display list. From Tutorial #2, we know that a display must contain the following:

1. Three blank lines

2. A Load Memory Scan byte. This tells ANTIC what graphics mode to start print, and the following two bytes inform ANTIC of the beginning of screen RAM.

3. Mode lines for any combination of graphics modes, as long as these mode lines add up to a total of exactly 192 scan lines.

4. A jump instruction to let ANTIC know where the beginning of the Display List is. The following two bytes are the low and high byte of the DL location.

We think we can see now the function of the Display List. First ANTIC prints 3 blanks at the top of the screen, then it starts printing the contents of screen RAM using the stipulated graphics modes until it finds a jump instruction.

Then it jumps back to the beginning of the display list and starts the process all over again. Here's how we accomplish the same thing from machine language. We'll put the display list at the beginning of page 6 (1536 decimal), and our 6502 routine at $660 (1632 decimal).

```
0000           10             *=    $0660
006A           20 RAMTOP      =     $6A
0600           30 DL          =     $0600
0230           40 DLVECT      =     $230
00CE           50 SCREEN      =     $CE
0058           60 SCRN        =     $58
0660 A900      70             LDA   #$00
0662 A8        80             TAY
0663 85CE      90             STA   SCREEN
0665 8558      0100           STA   SCRN
0667 A56A      0110           LDA   RAMTOP
0669 38        0120           SEC
066A E910      0130           SBC   #$10
066C 85CF      0140           STA   SCREEN+1
066E 8559      0150           STA   SCRN+1
0670 A970      0160           LDA   #$70
0672 990006    0170 BLANK     STA   DL,Y
0675 C8        0180           INY
0676 C003      0190           CPY   #$03
0678 D0F8      0200           BNE   BLANK
067A A94D      0210           LDA   #$4D
067C 990006    0220           STA   DL,Y
067F C8        0230           INY
0680 A5CE      0240           LDA   SCREEN
0682 990006    0250           STA   DL,Y
0685 C8        0260           INY
0686 A5CF      0270           LDA   SCREEN+1
0688 990006    0280           STA   DL,Y
068B C8        0290           INY
```

Let's explain the variables we use. RAMTOP is the OS pointer to the top of memory. We have to reserve about 4K for our screen (72 mode lines of GRAPHICS7 [ANTIC mode 13], and 3 mode lines of GRAPHICS 2 [ANTIC mode 6]), so we must subtract RAMTOP. DL is the location of our Display List, and DLVECT is the OS vector to the location of our Display List. We must change this to let OS know where to start looking for our homebrew DL. SCRN is the OS pointer which points to the beginning of screen RAM. We must change this pointer to point to our protected RAM. SCREEN is our software pointer pointing to the beginning of the screen.

In lines 70-150, we subtract 4K (16*256) from RAMTOP and store this address in both our software pointer and the OS pointer.

Then we load the accumulator with the $70 (112 decimal), which is the ANTIC command for "print a blank line". Our blank loop prints 3 of these lines (lines 170-200). Now we must add the LMS byte to our DL. By definition, the LMS byte must equal 64 + the ANTIC mode we wish to start printing in (64 + 13 = 77) $4D. We store this in our Display List. Then we store the low and high byte of screen RAM in the DL (lines 240-280).

```
068C A90D      0300 GRPHC7 LDA   #$0D
068E 990006    0310        STA   DL,Y
```

```
0691 C8      0320        INY
0692 C04E    0330        CPY  #$4E
0694 D0F6    0340        BNE  GRPHC7
```

Next, we insert a loop to print 72 mode lines of GRAPHICS 7, ANTIC mode $OD to the Display List.

```
0696 A906    0350 GRPHC2 LDA  #$06
0698 990006  0360        STA  DL,Y
069B C8      0370        INY
069C C051    0380        CPY  #$51
069E D0F6    0390        BNE  GRPHC2
```

Then, we'll add 3 lines of GRAPHICS mode Z. This gives us a total of 192 scan lines:

mode lines	scanlines per mode line	total
72	2	144
3	16	48
		192 scan lines

```
06A0 A941    0400        LDA  #$41
06A2 990006  0410        STA  DL,Y
06A5 C8      0420        INY
06A6 ADC106  0430        LDA  DLIST
06A9 990006  0440        STA  DL,Y
06AC C8      0450        INY
06AD ADC206  0460        LDA  DLIST+1
06B0 990006  0470        STA  DL,Y
```

After letting ANTIC know what we want our screen to look like, we must tell it where to jump so it can restart the process of printing the screen using our display list. So we add the jump instruction $41, 65 decimal to the Display List, and then the low and high byte of the location of our Display List.

Now our display list is done. However, we must initialize the process by letting the OS know where our new Display List is.

```
06B3 ADC106  0480        LDA  DLIST
06B6 8D3002  0490        STA  DLVECT
06B9 ADC206  0500        LDA  DLIST+1
06BC 8D3102  0510        STA  DLVECT+1
06BF 68      0520        PLA
06C0 60      0530        RTS
06C1 0006    0540 DLIST  .WORD DL
```

So last, but not least, we load the accumulator with the low and high byte of the location of our Display List, and store this in the OS Display List location vector.

Now that we've set up our display list, we're ready to begin writing our shape tables and bit mapping routine. First let's see how to set up our shape table. Our shape table consists of two shapes. The first two bytes of each shape holds the length and width of the shape. The rest of the bytes contain length x width bytes which define our shape. Simple, right? Let's look at the bit mapping routine.

```
0000        10          *=    $4000
00CB        20 SHNUM    =     $CB
00CC        30 XPOS     =     $CC
00CD        40 YPOS     =     $CD
00CE        50 IND1     =     $CE
00D0        60 IND2     =     $D0
00DA        70 LENGTH   =     $DA
00DB        80 WIDTH    =     $DB
0058        90 SCREEN   =     $58
4000 48     0100        PHA
4001 98     0110        TYA
4002 48     0120        PHA
4003 8A     0130        TXA
4004 48     0140        PHA
4005 08     0150        PHP
4006 D8     0160        CLD
```

SHNUM is our shape number pointer. It points to the shape we want to draw on the screen. XPOS is the horizontal position where we want to draw our shape. Since our screen is 40 bytes wide, valid variables range from 0-39. YPOS holds the vertical location where we want to print our shape. Valid values range from 0-191. IND1 is a two byte pointer which points to the current shape table. IND2 is another 2 byte pointer which points to the screen location where the shape will be drawn. LENGTH is the length of the current shape, and WIDTH is the width of the shape. SCREEN is the OS vector pointing to the beginning of screen RAM. Remember that we have already changed SCREEN to point to our homemade screen.

Since we are using this routine from BASIC, we must ensure that we are not altering any registers used by BASIC. Thus, we push the contents of the accumulator, X-Register, Y-Register, and the Processor Status Register onto the stack. Then we clear the decimal mode. Now we're ready to start printing shapes.

```
4007 A5CB   0170        LDA   SHNUM
4009 0A     0180        ASL   A
400A AA     0190        TAX
400B BD6006 0200        LDA   SHTAB,X
400E 85CE   0210        STA   IND1
4010 BD6106 0220        LDA   SHTAB+1,X
4013 85CF   0230        STA   IND1+1
4015 A558   0240        LDA   SCREEN
4017 85D0   0250        STA   IND2
4019 A559   0260        LDA   SCREEN+1
401B 85D1   0270        STA   IND2+1
```

First, we load the accumulator with the shape number we want to draw. Since the address of the shape is stored as a two byte pointer, we must multiply the shape number by 2 to access this address from the shape table. Then we load the accumulator with the low and high bytes of the shape table address, and store this address in IND1. Next, we store the location of the beginning of screen RAM in IND2.

```
401D A4CD    0280           LDY   YPOS
401F F010    0290           BEQ   NOY
4021 A5D0    0300 ADLP1     LDA   IND2
4023 18      0310           CLC
4024 6928    0320           ADC   #$28
4026 85D0    0330           STA   IND2
4028 A5D1    0340           LDA   IND2+1
402A 6900    0350           ADC   #$00
402C 85D1    0360           STA   IND2+1
402E 88      0370           DEY
402F D0F0    0380           BNE   ADLP1
```

Here we find the vertical location where we want to draw our shape. First we check the value of YPOS. If it's zero, we know that the shape is to be drawn at the top of the screen, and branch to check for the horizontal location (NOY). Otherwise, we must add $28 (40 decimal) to IND2 for each unit of YPOS. We do this because we know that each line is 40 bytes long. To access each line, we must add 40 bytes.

```
4031 A5D0    0390 NOY       LDA   IND2
4033 18      0400           CLC
4034 65CC    0410           ADC   XPOS
4036 85D0    0420           STA   IND2
4038 A5D1    0430           LDA   IND2+1
403A 6900    0440           ADC   #$00
403C 85D1    0450           STA   IND2+1
```

Now we want to add the horizontal location of our shape to IND2.

```
403E B1CE    0460           LDA   (IND1),Y
4040 85DA    0470           STA   LENGTH
4042 C8      0480           INY
4043 B1CE    0490           LDA   (IND1),Y
4045 85DB    0500           STA   WIDTH
4047 A5CE    0510           LDA   IND1
4049 18      0520           CLC
404A 6902    0530           ADC   #$02
404C 85CE    0540           STA   IND1
404E A5CF    0550           LDA   IND1+1
4050 6900    0560           ADC   #$00
4052 85CF    0570           STA   IND1+1
```

Here we take the first 2 bytes of our shape table and store them in LENGTH and WIDTH. Then we update our shape table pointer to bypass these first two bytes. IND1 will now point to the first byte of our shape description.

```
4054 A4DB    0580 AGAIN     LDY   WIDTH
4056 B1CE    0590 STLOOP    LDA   (IND1),Y
4058 91D0    0600           STA   (IND2),Y
405A 88      0610           DEY
405B D0F9    0620           BNE   STLOOP
405D A5CE    0630           LDA   IND1
405F 18      0640           CLC
```

```
4060 65DB    0650          ADC  WIDTH
4062 85CE    0660          STA  IND1
4064 A5CF    0670          LDA  IND1+1
4066 6900    0680          ADC  #$00
4068 85CF    0690          STA  IND1+1
406A A5D0    0700          LDA  IND2
406C 18      0710          CLC
406D 6928    0720          ADC  #$28
406F 85D0    0730          STA  IND2
4071 A5D1    0740          LDA  IND2+1
4073 6900    0750          ADC  #$00
4075 85D1    0760          STA  IND2+1
4077 C6DA    0770          DEC  LENGTH
4079 D0D9    0780          BNE  AGAIN
```

We're ready to start printing our shape on the screen. We load the Y-register with the width of the shape (left to right), and load from the shape table (IND1), and store to the screen until one line of the shape is drawn. Next, we update IND1 to the next line of shape table data. Then we add $28, 40 decimal, to our screen pointer, to access the next line of the screen. We decrement the LENGTH variable and continue printing until the shape is drawn.

```
407B 28      0790          PLP
407C 68      0800          PLA
407D AA      0810          TAX
407E 68      0820          PLA
407F A8      0830          TAY
4080 68      0840          PLA
4081 68      0850          PLA
4082 60      0860          RTS
```

Finally, we restore the registers and return to BASIC.

```
4083          0870          *=    $660
0660 6406     0880 SHTAB    .WORD SH0,SH1
0662 A606
0664 08       0890 SH0      .BYTE $08,$08
0665 08
0666 FF       0900          .BYTE $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
0667 FF
0668 FF
0669 FF
066A FF
066B FF
066C FF
066D FF
066E FF       0910          .BYTE $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
066F FF
0670 FF
0671 FF
```

```
0672 FF
0673 FF
0674 FF
0675 FF
0676 FF        0920          .BYTE $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
0677 FF
0678 FF
0679 FF
067A FF
067B FF
067C FF
067D FF
067E FF        0930          .BYTE $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
067F FF
0680 FF
0681 FF
0682 FF
0683 FF
0684 FF
0685 FF
0686 FF        0940          .BYTE $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
0687 FF
0688 FF
0689 FF
068A FF
068B FF
068C FF
068D FF
068E FF        0950          .BYTE $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
068F FF
0690 FF
0691 FF
0692 FF
0693 FF
0694 FF
0695 FF
0696 FF        0960          .BYTE $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
0697 FF
0698 FF
0699 FF
069A FF
069B FF
069C FF
069D FF
069E FF        0970          .BYTE $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
069F FF
06A0 FF
06A1 FF
06A2 FF
06A3 FF
06A4 FF
06A5 FF
```

```
06A6 08       0980 SH1     .BYTE $08,$08
06A7 08
06A8 00       0990         .BYTE $00,$00,$00,$00,$00,$00,$00,$00
06A9 00
06AA 00
06AB 00
06AC 00
06AD 00
06AE 00
06AF 00
06B0 00       1000         .BYTE $00,$00,$00,$00,$00,$00,$00,$00
06B1 00
06B2 00
06B3 00
06B4 00
06B5 00
06B6 00
06B7 00
06B8 00       1010         .BYTE $00,$00,$00,$00,$00,$00,$00,$00
06B9 00
06BA 00
06BB 00
06BC 00
06BD 00
06BE 00
06BF 00
06C0 00       1020         .BYTE $00,$00,$00,$00,$00,$00,$00,$00
06C1 00
06C2 00
06C3 00
06C4 00
06C5 00
06C6 00
06C7 00
06C8 00       1030         .BYTE $00,$00,$00,$00,$00,$00,$00,$00
06C9 00
06CA 00
06CB 00
06CC 00
06CD 00
06CE 00
06CF 00
06D0 00       1040         .BYTE $00,$00,$00,$00,$00,$00,$00,$00
06D1 00
06D2 00
06D3 00
06D4 00
06D5 00
06D6 00
06D7 00
06D8 00       1050         .BYTE $00,$00,$00,$00,$00,$00,$00,$00
06D9 00
```

```
06DA 00
06DB 00
06DC 00
06DD 00
06DE 00
06DF 00
06E0 00      1060          .BYTE $00,$00,$00,$00,$00,$00,$00,$00
06E1 00
06E2 00
06E3 00
06E4 00
06E5 00
06E6 00
06E7 00
```

Next, load in our machine language tutorial program and type RUN. Then hit [RETURN]. Our shape will move from left to right then down. The reason why it blinks is because we are printing two shapes. One is a box, and the next is a blank shape used to erase the shape before printing a new one. Since we're still moving the shape from BASIC (list lines 1050-1070), it's slow enough to give a blinking effect.

Now you have almost everything you need to write your own game, except for collision detection and some sound effects. We'll explore these in our next issue.

Utility of the Month

Our utility of the month allows you to change the variable names in your programs. Now, if after writing a program, you wish to change the names of variables to less descriptive (or more descriptive) names, it's easily done. Once you have loaded in the program, type X=USR(39199) to activate.

Pharoah's Curse Adventure

Your expedition in the Sahara has stumbled across the burial tomb of an unknown Pharoah. The seals on the door have been broken, and hieroglyphs just inside the door speak of golden tablets buried with the Pharoah, and warn of a terrible curse.

The objective is to go into the tomb, find the gold tablets, if possible, and escape the curse of the Pharoah (with the tablets).

When you enter a room, you will be given a description of the room, and asked for a command. Some commands are as follows:

LEAVE -- After this command, you will be prompted with "DIRECTION", to which you press N, S, E, or W. Alternatively, the computer recognizes the commands N, S, E, and W as one-letter words.

GET -- To pick up an object.

INVENT -- To get a list of what you are carrying.

SEARCH -- To gain additional information about an object.

READ -- To read an object. You can understand much of the hieroglyphs in the tomb.

It is up to you to figure out the untold commands needed to find the gold tablets.

Except for "LEAVE", "INVENT", and the one-letter commands "N", "S", "E", and "W", all commands are two words long. For example: GET BOX.

Remember that dropped items don't necessarily remain where they were dropped.

Disassembler

DISASM is a 6502 disassembler, written in BASIC. It can work in decimal or hexadecimal, with output to screen or printer.

A "disassembler" is a program which translates machine language code to its alphabetic "mnemonic", or assembly language code. Thus, meaningless numbers become readable program lines.

In order to use this disassembler, you must first be able to understand 6502 assembly language. Once you've crossed that mountain, you're on your way to taking apart the BASIC, ROM, Operating System ROM, cartridge, or machine language subroutines.

When you run the program, the first prompt will ask you which system you want to work in - decimal or hexadecimal (base 16). Press "D" or "H" for your choice. The only thing to note here is that disassembly in hex is slightly slower than in decimal. In any case, you may change modes at any time during the program. Remember, though, that once you are in a mode, all of your responses must be in that mode. Thus, at the next prompt, "Starting Address?", if you are in decimal you might answer 1536, while in hex you would have to answer 0600.

After returning a starting address, you'll see the first page of the disassembly. The output follows this structure:

1. Address
2. Op Code byte plus 0, 1, or 2 operand bytes
3. ASCII representation, if possible, of those bytes
4. Op Code Mnemonic
5. Operand Mnemonic

Note that 4 and 5 are the actual disassembled code. Sample output:

```
Address     Bytes     ASCII     Mnemonic     Operand
 $0600                            LDA         $0600
 $0603                            RTS
```

Two symbols you may see are:

$ - means hex numbers

- immediate addressing mode

After 20 lines of code are printed, you will be asked for a "Command?". There are four possible responses:

<space> - continue disassembly from current address
M - change number mode (decimal to hex or hex to decimal)
<return> - define new starting address
P - disassemble to printer

For the last one, P, there are two prompts. First you give a starting address, and then an ending address at which to stop the disassembly. Output structure is the same as appears on the screen.

The following tables are addresses in the BASIC ROM and the Operating System (OS) ROM to help you in disassembling them:

BASIC ROM

40960 - 41036	Cold Start
41037 - 41055	Warm Start
41056 - 42081	Syntax
42082 - 42158	Search
42159 - 42508	Statement Name Table
42509 - 43134	Syntax Tables
43135 - 43358	Memory Manager

43359 - 43519	Execute CONT
43520 - 43631	Statement Table
43632 - 43743	Operator Table
43744 - 44094	Execute Expression
44095 - 44163	Operator Precedence
44164 - 45001	Execute Operation
45002 - 45320	Execute Function
45321 - 47127	Execute Statement
47128 - 47381	CONT Subroutines
47382 - 47542	Errors
47543 - 47732	Graphics
47733 - 48548	I/O Routines
48549 - 49151	Floating Point

OPERATING SYSTEM ROM

55296 - 57343	Floating Point Routines
57344 - 58367	Character Set
58368 - 58533	Vectors (see below)
58534 - 59092	CIO
59093 - 59715	Interrupt Handler
59716 - 60905	SIO
60906 - 61047	Disk Handler
61048 - 61248	Printer Handler
61249 - 61666	Cassette Handler
61667 - 62435	Monitor
62436 - 65535	Display and Keyboard Handler

OS VECTORS

58368 - 58383	Editor
58384 - 58399	Screen
58400 - 58415	Keyboard
58416 - 58431	Printer
58432 - 58447	Cassette
58448 - 58495	Jump Vectors
58496 - 58533	Initial RAM Vectors

Left cartridge (BASIC) begins at 40960.
Right cartridge (when present) begins at 32768.

Good Luck!

End Note

Both the DATABOSS and DATAPRINT are intricately written programs. If you should have any problems with either the DATABOSS or the DATAPRINT programs please let us know immediately and we will print the corrections in the next issue of Magatar.

To report a problem, please jot down the following information:

1) The error number.

2) The line number in which the error was found.
3) What you were doing at the time of the error.
4) Any file specifications that might apply.

Thanks for your help in this matter.